Formal Analysis of Control Software for Cyber-Physical Systems

Peter Herrmann
Norwegian University of Science and Technology (NTNU), Trondheim, Norway
Email: herrmann@ntnu.no

Jan Olaf Blech
RMIT University
Melbourne, Australia
Email: joblech@gmail.com

***Abstract*—Modern Cyber-Physical Systems are often driven by a plethora of controllers that are connected with each other and their environment. To guarantee a safe and robust execution of the systems, their control units have to strictly fulfill certain properties which calls for the use of formal analysis methods in the software development process. We present the combination of the model-based engineering technique Reactive Blocks and the spatiotemporal analysis tool BeSpaceD facilitating the formal verification of controller software.**

I. Introduction

Cyber-Physical Systems (CPS) play an increasing role in several technical domains. One of these areas is the transport sector in which autonomy of vehicles and their automatic coordination are an important part of the agenda. Modern cars are equipped by hundreds of Electronic Control Units (ECU) that manage relevant functions [1]. These ECUs are connected with each other and more or more also with other vehicles and fixed stations forming so-called Cooperative Intelligent Transport Systems (C-ITS) [2].

Of course, CPSs have to fulfill very strict requirements with respect to safety, robustness, and reliability since any malfunctioning may endanger the systems themselves and humans in them or their vicinity. Therefore, the control software has to be intensively tested for both, being functionally correct and fulfilling certain Quality of Service (QoS) properties. A good companion to classical testing is formal analysis of control software that guarantees that software errors are already detected in the engineering phase. To facilitate the formal-based development of controllers, we have combined the two tool-sets Reactive Blocks [3] and BeSpaceD [4] which will be discussed in the rest of the paper.

II. Reactive Blocks and BeSpaceD

The main concept of the model-based engineering technique Reactive Blocks is the *building block* that allows us to model sub-functionality in separation [3]. This has the advantage that recurring functions can be specified once in a building block, stored in a library of the tool, and easily reused in any system model that needs this functionality.

In Fig. 1, we show a building block for an example in industrial automation (see [5]). Here, system behavior is realized by UML activity graphs that offer an easily understandable flow semantics close to Petri nets. The example consists of

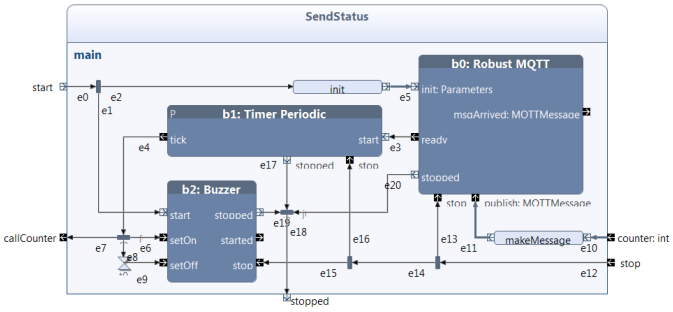

Fig. 1. An example block (taken from [5])

three inner building blocks *Timer Periodic*, *Buzzer*, and *Robust MQTT* taken from the libraries. They are linked by edges and typical UML activity nodes such as forks or timers defining the control and data flows in the system [6]. In operations like *makeMessage*, we add Java methods that are carried out when a flow passes an operation.

The interface of a building block is realized by pins resp. parameter nodes. For instance, a flow may head from our block *SendStatus* to the inner block *Robust MQTT* through the pin *init* of the inner block. At the edge of its activity, *Robust MQTT* has a parameter node of the same name through which the flow continues. Moreover, we use so-called External State Machines (ESM) [7]. They describe in which execution state a certain parameter node may be passed by a flow.

A building block has to realize a behavior fulfilling both, its own ESM and the ones of its inner blocks. We provided the UML activities and ESMs with formal semantics [6]. This allowed us to integrate a model checker into the tool verifying whether the ESMs are correctly realized and the system fulfills other properties such as freedom of deadlocks [3]. The UML models are automatically transformed into efficiently executable Java code [8].

The verification tool BeSpaceD enables us to solve constraints and to carry out non-classical model checking particularly for spatiotemporal systems [4]. For that, it offers a modeling language based on abstract data types and a library to reason on models, e.g., by state-space exploration, abstraction, or reduction. Further, BeSpaceD makes it possible to create verification goals for SAT and SMT solvers.

BeSpaceD was developed in Scala making it compatible

Fig. 2. The methodology (taken from [12])

with the Eclipse-based Reactive Blocks. Moreover, Scala can be used to create models that are based on abstract data types. This allows us to use SAT and SMT for much more concrete models than their traditional inputs. BeSpaceD is typically used to verify if a moving vehicle can come into conflict with an obstacle since it might react too late on a corresponding sensor input [9]. It can be used both at design and runtime [10].

III. Tool Composition

Reactive Blocks and BeSpaceD can be combined in two different ways. In the first one, a model in Reactive Blocks is supplemented with a simulator of the technical process to be controlled [9]. Then, one carries out several simulations and converts the log data into a BeSpaceD input formula. This approach can be relatively easily used for many Reactive Blocks models, and BeSpaceD can prove the log data efficiently. The analysis, however, is not exhaustive since only the results of the simulation runs are effectively checked by BeSpaceD.

In the other composition type, the Reactive Blocks model is checked for the presence of particular system functionality from which descriptive formulas are extracted and converted into BeSpaceD formulas [11]. This allows us to verify spatiotemporal properties exhaustively but, in order to make the automatic formula extraction possible, the Reactive Blocks model has to use a certain set of building blocks. This takes freedom away from the software engineer which aggravates the development process.

In [12], we present a methodology for the model-based development of control software using the two tool-sets. It takes into account that one can only create controllers with in-depth knowledge about central kinematic properties of the system to be driven. This information, however, can often only be gathered by testing real prototypes. That is reflected by our methodology that uses the five steps listed in Fig. 2.

In step 1, one develops an initial version of the control software that only contains tentative safety mechanisms but allows us to test the prototype which happens in step 2. After having learned the relevant kinematic parameters, the original control software is extended in step 3 by functionality making the system resilient and robust. This can be nicely done in Reactive Blocks by simply extending the original model with building blocks that realize the functions to be adjoined. In step 4, we analyze the amended model with BeSpaceD using one of the two combination mechanisms explained above. If all spatiotemporal verifications succeed, we generate the executable code in step 5.

Our methodology is not seen as a replacement for traditional certification but as a supplement. The quality of the developed software should be better than with traditional programming making the certification process much easier.

IV. Conclusion

We discussed how one can combine Reactive Blocks and BeSpaceD for the development of control software during design time. In the next step, we plan also to utilize the ability of BeSpaceD to be executed during runtime [10]. Particularly, we will realize BeSpaceD execution environments in building blocks that can be easily added to control software models. The controllers can then simply trigger the verification of certain spatiotemporal properties and directly react on the results of these checks. A first sketch of using this for the decision about reconfiguring controllers is presented in [13].

References

[1] C. Valasek and C. Miller, "Car Hacking: The Content," http://blog.ioactive.com/2013/08/, 2013, accessed: 2017-06-23.

[2] A. Festag, "Cooperative Intelligent Transport Systems Standards in Europe," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 166–172, 2014.

[3] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2068–2080, 2009.

[4] J. O. Blech and H. Schmidt, "Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems," in *Improving Systems and Software Engineering Conference*, 2013.

[5] P. Herrmann and J. O. Blech, "Formal Model-based Development in Industrial Automation with Reactive Blocks," in *3rd Human-Oriented Formal Methods Workshop (HOFM2016)*, Vienna, July 2016.

[6] F. A. Kraemer and P. Herrmann, "Reactive Semantics for Distributed UML Activities," in *Joint WG6.1 International Conference (FMOODS) and WG6.1 International Conference (FORTE)*, ser. LNCS 6117. Springer-Verlag, 2010, pp. 17–31.

[7] ——, "Automated Encapsulation of UML Activities for Incremental Development and Verification," in *Model Driven Engineering Languages and Systems (MoDELS)*, ser. LNCS 5795. Springer-Verlag, 2009, pp. 571–585.

[8] F. A. Kraemer, P. Herrmann, and R. Bræk, "Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services," in *8th International Symposium on Distributed Objects and Applications (DOA06)*, ser. LNCS 4276. Springer-Verlag, 2006, pp. 1614–1632.

[9] F. Han, J. O. Blech, P. Herrmann, and H. Schmidt, "Towards Verifying Safety Properties of Real-Time Probability Systems," in *11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA)*. EPTCS, 2014.

[10] J. O. Blech, L. Fernando, K. Foster, Abhilash G, and Sudarsan SD, "Spatio-temporal Reasoning and Decision Support for Smart Energy Systems," in *21st Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE Computer, 2016.

[11] P. Herrmann, J. O. Blech, F. Han, and H. Schmidt, "A Model-based Toolchain to Verify Spatial Behavior of Cyber-Physical Systems," *International Journal of Web Services Research (IJWSR)*, vol. 13, no. 1, pp. 40–52, 2016.

[12] S. Hordvik, K. Øseth, H. H. Svendsen, J. O. Blech, and P. Herrmann, "Model-based Engineering and Spatiotemporal Analysis of Transport Systems," in *Evaluation of Novel Approaches to Software Engineering*, ser. CCIS 703. Springer-Verlag, 2017, pp. 44–65.

[13] A. Taherkordi, P. Herrmann, J. O. Blech, and Á. Férnandez, "Service Virtualization for Self-Adaptation in Mobile Cyber-Physical Systems," in *Management of Service-Oriented Cyber-Physical Systems (MCPS)*, Banff, Canada, 2016.